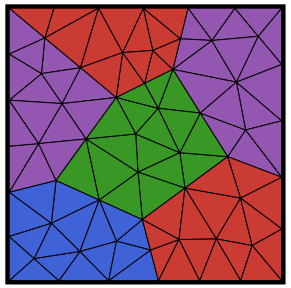
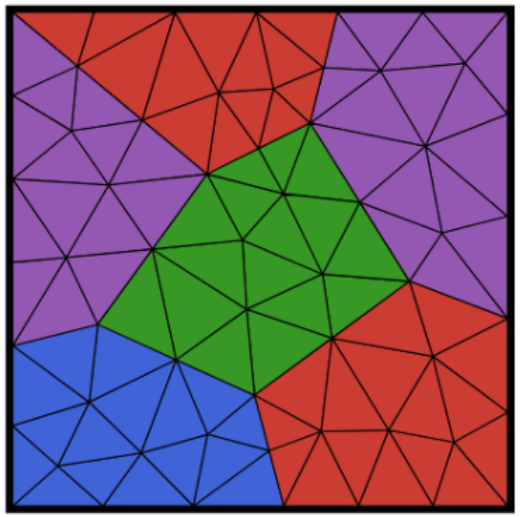


Introduction to the Finite Element Toolbox



Ferrite.jl

Kim Louisa Auth

Section of Solid Mechanics

DTU

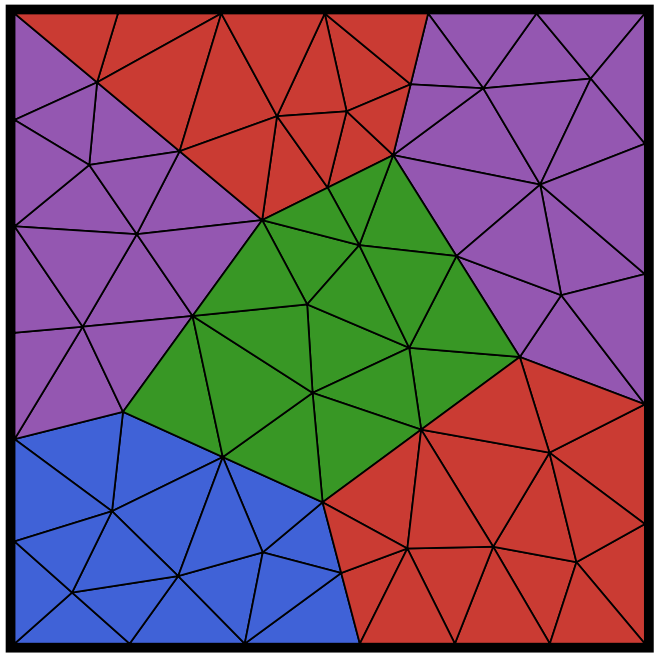
Co-authors:

Knut Andreas Meyer

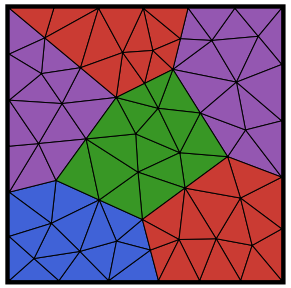
Fredrik Ekre

Outline

- What is the Julia Programming Language?
- History and users of Ferrite.jl
- What is Ferrite?
  - The FEM Puzzle Pieces
- **How to use Ferrite?**
- Research examples
- Concluding remarks and questions

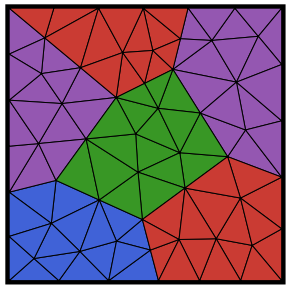


What is Julia?



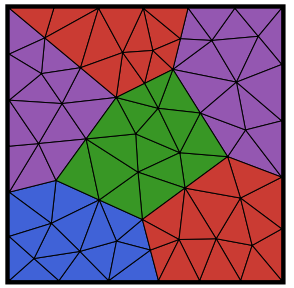
- Open Source (MIT License)
- A young language
  - First public in 2012
  - Release 1.0 in 2018
  - Currently 1.11.6
- “Easy as python”
  - High-level dynamic programing language
- “Fast as C”
  - Just-in-time compilation

History and users of Ferrite.jl



- **Kristoffer Carlsson** & **Fredrik Ekre**, Chalmers (2016)
- A **toolbox** providing FE building blocks:
  - *Inspired by Deal.ii*
- **Adoption**
  - 37 different contributors
  - 394 github stars / 166 members in Slack
- **FerriteCon**
  - 2022: Braunschweig, Germany
  - 2023: Bochum, Germany
  - 2024: Gothenburg, Sweden
  - 2025: Copenhagen, Denmark
- Version 1.0 released in 2024

FEM Puzzle Pieces: What does Ferrite provide?



1) Geometry
2) Mesh

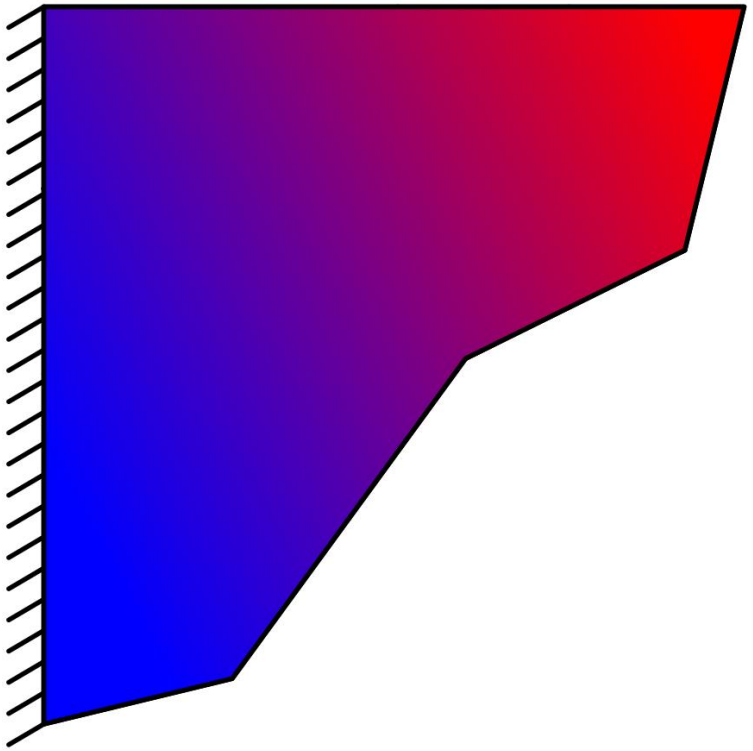
- Simple builtin mesh generator
- `Gmsh` interface package `FerriteGmsh.jl`
- Abaqus input file parser `FerriteMeshParser.jl`

3) **Degrees of Fr**
   - Vector field, **u**
   - Scalar field, p (only purple domain)
4) **Tools for assembly**
   - Shape functions evaluation (interpolations)
   - Mapping shape functions to cell coordinates
   - Quadrature rules
   - Assemble local into to global sparse matrix
5) **Constraint handling**
6) Linear solver

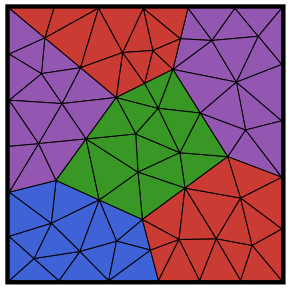
- **"W"**
- **LinearSolve.jl**

7) Post-processing
   - **Quadrature point data (*L2Projector*)**
   - Visualization

- Builtin VTK (ParaView) export
- **FerriteViz.jl**



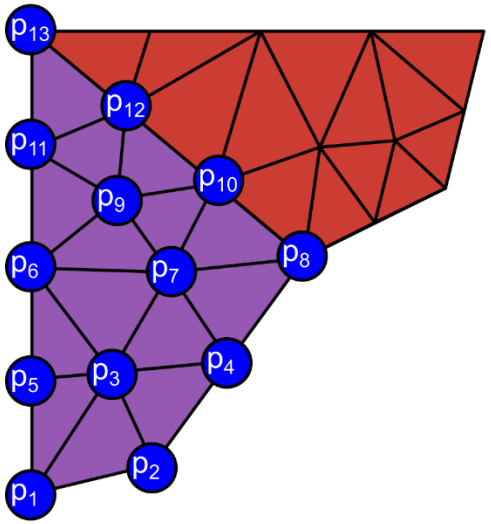
FEM Puzzle Pieces: How does Ferrite do that?

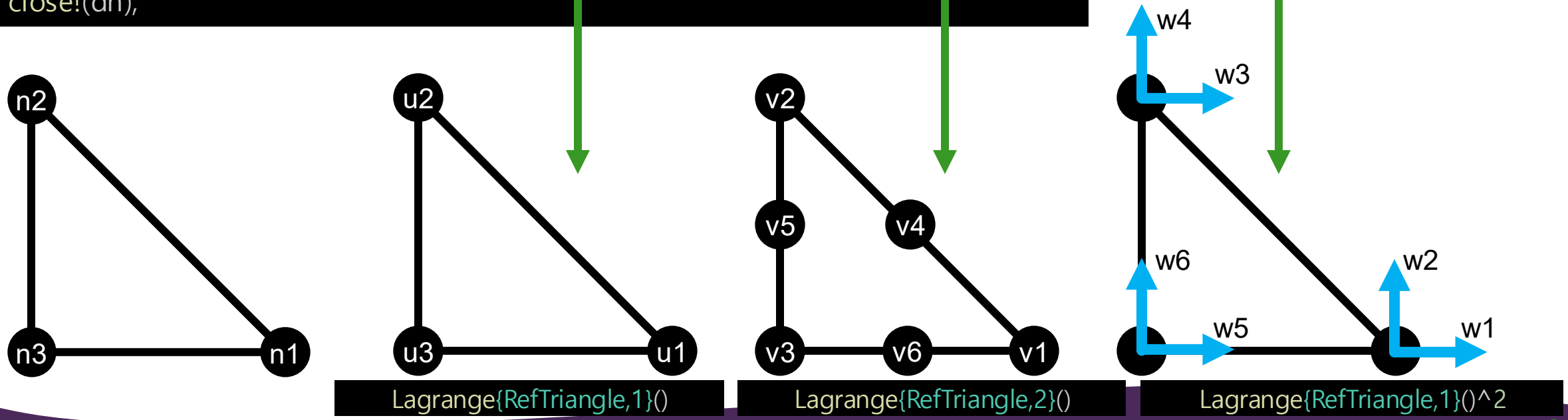


1) Tensors.jl
2) Stationary heat equation (Poisson's equation)
3) Triangle to Quadrilateral & 2D -> 3D
4) Change to linear elasticity
5) Advanced setup
   - *Porous media with solid aggregates*
   - *Mixed element shapes*
6) Advanced showcase
   Navier-Stokes with DifferentialEquations.jl
7) *Research examples*

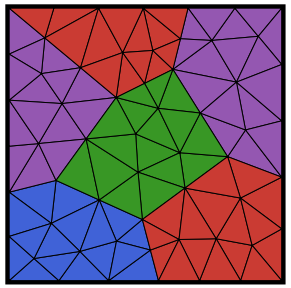
DoF Distribution

```
grid = generate_grid(Triangle, (20, 20));
ip   = Lagrange{RefTriangle, 1}()
ip2  = Lagrange{RefTriangle, 2}()

dh = DofHandler(grid)
add!(dh, :u, ip)
add!(dh, :v, ip2)
add!(dh, :w, ip^2)
close!(dh);
```





Assembly: Calculate cell contribution



Setup

```
ip = Lagrange{RefTriangle, 1}()          # Defined before
ip_geo = ip                             # Geometric interpolation
qr = QuadratureRule{RefTriangle}(2)      # Numerical integration
cellvalues = CellValues(qr, ip, ip_geo)  # "Shape function object"
```

Element routine

```
function element_routine!(Ke::Matrix, fe::Vector, cellvalues::CellValues, x::Vector)
    # Map shape functions etc. to current geometry
    reinit!(cellvalues, x)
    # Loop over quadrature points
    for q_point in 1:getnquadpoints(cellvalues)
        # Get the quadrature weight
        dΩ = getdetJdV(cellvalues, q_point)
        # Loop over test shape functions
        for i in 1:getnbasefunctions(cellvalues)
            δN  = shape_value(cellvalues, q_point, i)
            ∇δN = shape_gradient(cellvalues, q_point, i)
            # Add contribution to fe
            fe[i] += δN * dΩ
            # Loop over trial shape functions
            for j in 1:getnbasefunctions(cellvalues)
                ∇N = shape_gradient(cellvalues, q_point, j)
                # Add contribution to Ke
                Ke[i, j] += (∇δN ⋅ ∇N) * dΩ
            end
        end
    end
    return Ke, fe
end
```

Heat Equation

$$\int_\Omega \nabla \delta u \cdot \nabla u \, \mathrm{d}\Omega = \int_\Omega \delta u \, \mathrm{d}\Omega$$

$$\left[\int_\Omega \nabla \delta N_i \cdot \nabla N_j \, \mathrm{d}\Omega \right] a_j = \int_\Omega \delta N_i \, \mathrm{d}\Omega$$

$$K_{ij} a_j = f_i$$

```julia
function element_routine!(Ke::Matrix, fe::Vector, cellvalues::CellValues, cellcoords::Vector)
    # Map shape function gradients etc. to current geometry
    reinit!(cellvalues, cellcoords)
    # Loop over quadrature points
    for q_point in 1:getnquadpoints(cellvalues)
        # Get the quadrature weight
        dΩ = getdetJdV(cellvalues, q_point)
        # Loop over test shape functions
        for i in 1:getnbasefunctions(cellvalues)
            δNᵢ  = shape_value(cellvalues, q_point, i)
            ∇δNᵢ = shape_gradient(cellvalues, q_point, i)
            # Add heat source to fe
            fe[i] += δNᵢ * dΩ
            # Loop over trial shape functions
            for j in 1:getnbasefunctions(cellvalues)
                ∇Nⱼ = shape_gradient(cellvalues, q_point, j)
                # Add contribution to Ke
                Ke[i, j] += (∇δNᵢ ⋅ ∇Nⱼ) * dΩ
            end
        end
    end
    return Ke, fe
end
```

Heat Equation

$$\int_\Omega \nabla\delta u \cdot \nabla u \, \mathrm{d}\Omega = \int_\Omega \delta u \, \mathrm{d}\Omega$$

$$\left[\int_\Omega \nabla\delta N_i \cdot \nabla N_j \, \mathrm{d}\Omega\right] a_j = \int_\Omega \delta N_i \, \mathrm{d}\Omega$$

$$K_{ij} a_j = f_i$$

```julia
function element_routine!(Ke::Matrix, fe::Vector, cellvalues::CellValues, cellcoords::Vector)
    # Map shape function gradients etc. to current geometry
    reinit!(cellvalues, cellcoords)
    # Loop over quadrature points
    for q_point in 1:getnquadpoints(cellvalues)
        # Get the quadrature weight
        dΩ = getdetJdV(cellvalues, q_point)
        # Loop over test shape functions
        for i in 1:getnbasefunctions(cellvalues)
            δNᵢ  = shape_value(cellvalues, q_point, i)
            ∇δNᵢ = shape_gradient(cellvalues, q_point, i)
            # Add heat source to fe @ test function i
            fe[i] += δNᵢ * dΩ
            # Loop over trial shape functions
            for j in 1:getnbasefunctions(cellvalues)
                ∇Nⱼ = shape_gradient(cellvalues, q_point, j)
                # Add contribution to Ke @ test i, trial j
                Ke[i, j] += (∇δNᵢ ⋅ ∇Nⱼ) * dΩ
            end
        end
    end
    return Ke, fe
end
```

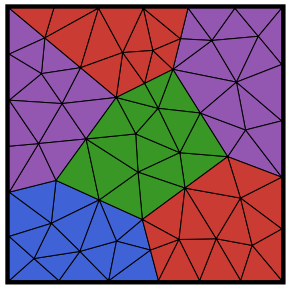
Heat Equation

$$\int_\Omega \nabla \delta u \cdot \nabla u \, \mathrm{d}\Omega = \int_\Omega \delta u \, \mathrm{d}\Omega$$

$$\left[\int_\Omega \nabla \delta N_i \cdot \nabla N_j \, \mathrm{d}\Omega\right] a_j = \int_\Omega \delta N_i \, \mathrm{d}\Omega$$

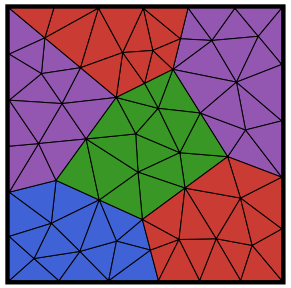
$$K_{ij} a_j = f_i$$

Assembly: Assemble cell contribution



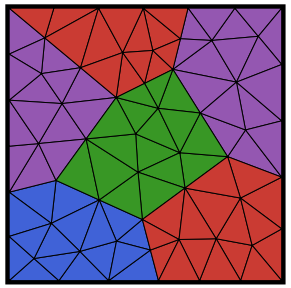
```
function assemble_global(cellvalues::CellValues, dh::DofHandler)
    # Allocate global stiffness matrix, K, and force vector, f
    K = allocate_matrix(dh)
    f = zeros(ndofs(dh))
    # Allocate the element stiffness matrix and element force vector
    n_basefuncs = getnbasefunctions(cellvalues)
    Ke = zeros(n_basefuncs, n_basefuncs)
    fe = zeros(n_basefuncs)

    assembler = start_assemble(K, f) # Create an assembler
    for cell in CellIterator(dh)     # Loop over all cells
        fill!(Ke, 0); fill!(fe, 0)   # Reset Ke and fe
        # Compute element contribution
        element_routine!(Ke, fe, cellvalues, getcoordinates(cell))
        # Assemble local, Ke and fe, into global, K and f
        assemble!(assembler, celldofs(cell), Ke, fe)
    end
    return K, f
end
```

Constraints: Dirichlet Boundary Conditions

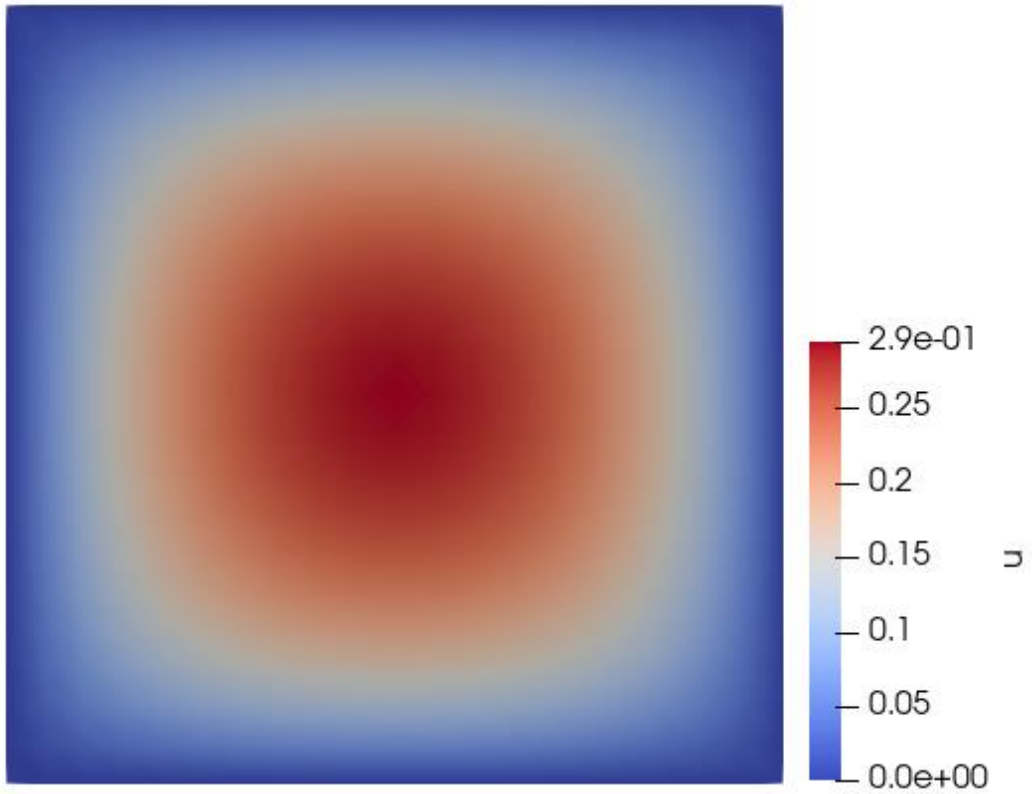


```
grid, dh, cellvalues = setup() # Pseudocode, see earlier slides

K, f = assemble_global(cellvalues, dh)

ch = ConstraintHandler(dh);

∂Ω = union(getfacetset(grid, "left"), getfacetset(grid, "right"),
           getfacetset(grid, "top"),  getfacetset(grid, "bottom"));

add!(ch, Dirichlet(:u, ∂Ω, (x, t) -> 0))

close!(ch)

apply!(K, f, ch) # Modify K and f to fulfill constraints
```

Putting it together: Stationary Heat Equation

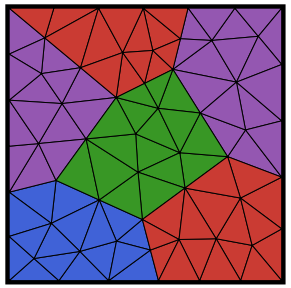


```
u = K \ f # Solve linear system

# Export solution
VTKGridFile("heat_equation", dh) do vtk
    write_solution(vtk, dh, u)
end
```



https://ferrite-fem.github.io/Ferrite.jl/dev/tutorials/heat_equation/

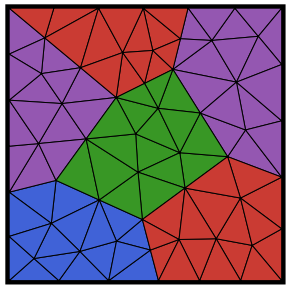
Change Triangles to Quadrilateral?



```
# Old setup
grid = generate_grid(Triangle, (20, 20));
ip   = Lagrange{RefTriangle, 1}()
qr   = QuadratureRule{RefTriangle}(2)
```

```
# New setup
grid = generate_grid(Quadrilateral, (20, 20));
ip   = Lagrange{RefQuadrilateral, 1}()
qr   = QuadratureRule{RefQuadrilateral}(2)
```

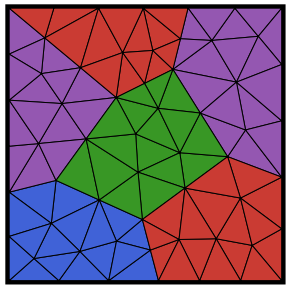
Change 2d to 3d?



```
# Old setup
grid = generate_grid(Triangle, (20, 20));
ip   = Lagrange{RefTriangle, 1}()
qr   = QuadratureRule{RefTriangle}(2)
```

```
# New setup
grid = generate_grid(Tetrahedron, (20, 20, 20));
ip   = Lagrange{RefTetrahedron, 1}()
qr   = QuadratureRule{RefTetrahedron}(2)
```

Change to linear elasticity



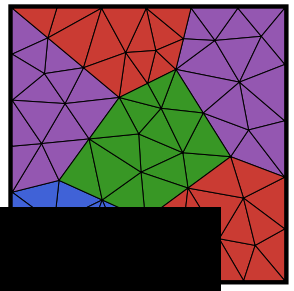
```
# Old setup
ip    = Lagrange{RefTriangle, 1}()
```

```
# New setup (vector problem in 2d)
ip    = Lagrange{RefTriangle, 1}()^2
```

+ new physics

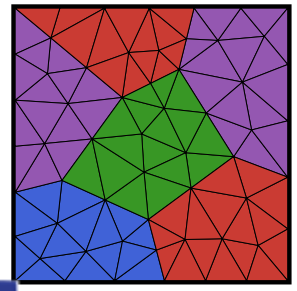
```
function element_routine!(Ke::Matrix, fe::Vector, cv::CellValues, cellcoords::Vector)
    reinit!(cv, cellcoords)                                # Map cellvalues to cell geometry
    for q_point in 1:getnquadpoints(cv)                    # Loop over quadrature points
        dΩ = getdetJdV(cv, q_point)                        # Get the quadrature weight
#=new=# C = calculate_stiffness(1.0, 2.0)                 # Stiffness from before
        for i in 1:getnbasefunctions(cv)                   # Loop over test shape functions
            δNᵢ  = shape_value(cv, q_point, i)
            ∇δNᵢ = shape_gradient(cv, q_point, i)
            fe[i] += δNᵢ * dΩ                              # Add heat source to fe
#=new=#      fe[i] += (δNᵢ · Vec((0.0, -1.0)))* dΩ         # Add body force to fe
            for j in 1:getnbasefunctions(cv)               # Loop over trial shape functions
                ∇Nⱼ = shape_gradient(cv, q_point, j)
                Ke[i, j] += (∇δNᵢ · ∇Nⱼ) * dΩ              # Add contribution to Ke
#=new=#          Ke[i, j] += (∇δNᵢ ⊡ C ⊡ ∇Nⱼ) * dΩ       # Add contribution to Ke
            end
        end
    end
    return Ke, fe
end
```

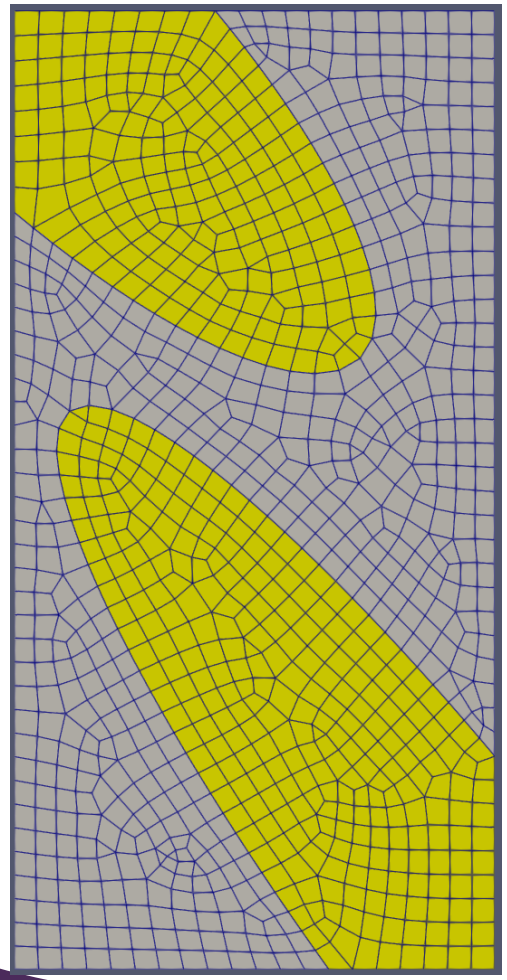
Change to linear elasticity



```
function element_routine!(Ke::Matrix, fe::Vector, cv::CellValues, cellcoords::Vector)
    reinit!(cv, cellcoords)                          # Map cellvalues to cell geometry
    for q_point in 1:getnquadpoints(cv)                # Loop over quadrature points
        dΩ = getdetJdV(cv, q_point)                   # Get the quadrature weight
#=new=# C = calculate_stiffness(1.0, 2.0)             # Stiffness from before
        for i in 1:getnbasefunctions(cv)              # Loop over test shape functions
            δNᵢ  = shape_value(cv, q_point, i)
            ∇δNᵢ = shape_gradient(cv, q_point, i)
            fe[i]# fe[i] += δNᵢ * dΩ
#=new=#      fe[i] += (δNᵢ ⋅ Vec((0.0, -1.0)))* dΩ     # Add body force to fe
            for j in 1:getnbasefunctions(cv)          # Loop over trial shape functions
                ∇Nⱼ = shape_gradient(cv, q_point, j)
                Ke[i, j]# Ke[i, j] += (∇δNᵢ ⋅ ∇Nⱼ) * dΩ
#=new=#          Ke[i, j] += (∇δNᵢ ⊡ C ⊡ ∇Nⱼ) * dΩ  # Add contribution to Ke
            end
        end
    end
    return Ke, fe
end
```

More advanced cases: Porous media, grid with mixed element shapes



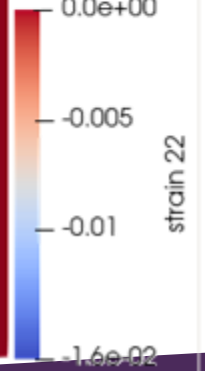


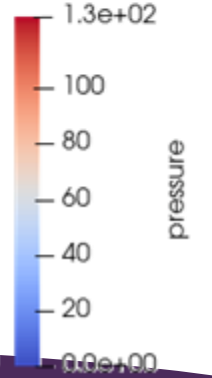
Solid aggregates

(Linear elasticity)

Porous matrix

(Linear poro-elasticity)

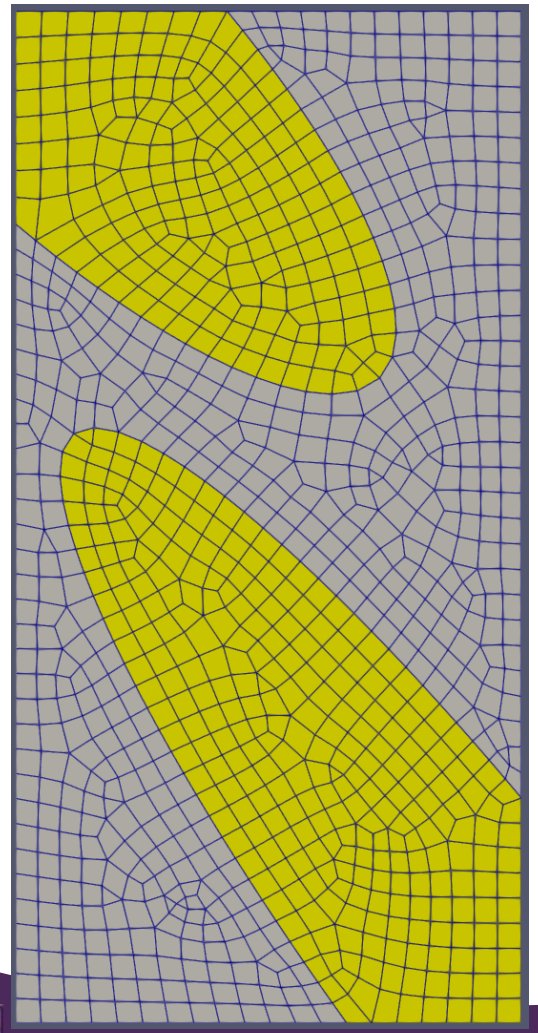




More advanced cases: Porous media, mixed grid

Solid aggregates
Linear elasticity
Displacement **u**(**x**)
Quads and triangles

Porous matrix
Linear poro-elasticity
Pressure p(**x**), and displacement **u**(**x**)
Quads and triangles

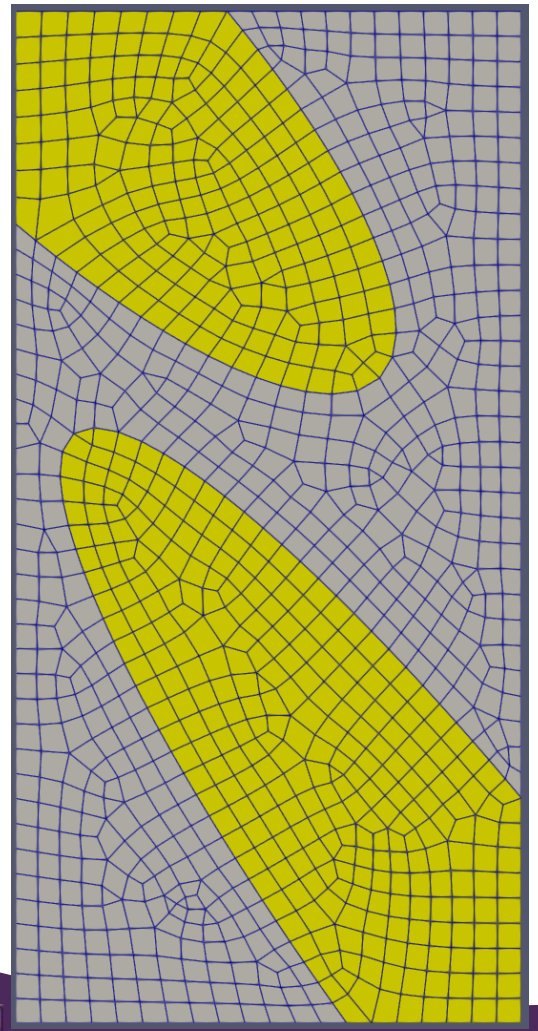


```
# Define interpolations
#  Quadratic displacement, quad elements
ipu_quad = Lagrange{RefQuadrilateral, 2}()^2
#  Quadratic displacement, triangular elements
ipu_tri  = Lagrange{RefTriangle, 2}()^2
#  Linear pressure, quad elements
ipp_quad = Lagrange{RefQuadrilateral, 1}()
#  Linear pressure, triangular elements
ipp_tri  = Lagrange{RefTriangle, 1}()

# Quadrature rules
qr_quad = QuadratureRule{RefQuadrilateral}(2) # 2x2 quadrature
qr_tri  = QuadratureRule{RefTriangle}(2)      # 3 quadrature points
```

More advanced cases: Porous media, mixed grid

Solid aggregates
Linear elasticity
Displacement **u**(**x**)
Quads and triangles

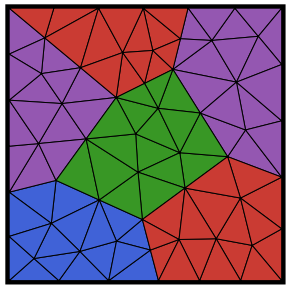
Porous matrix
Linear poro-elasticity
Pressure p(**x**), and displacement **u**(**x**)
Quads and triangles



```
# Setup the DofHandler
dh = DofHandler(grid)
# Solid quads
sdh_solid_quad = SubDofHandler(dh, getcellset(grid,"solid4"))
add!(sdh_solid_quad, :u, ipu_quad)
# Solid triangles
sdh_solid_tri = SubDofHandler(dh, getcellset(grid,"solid3"))
add!(sdh_solid_tri, :u, ipu_tri)
# Porous quads
sdh_porous_quad = SubDofHandler(dh, getcellset(grid, "porous4"))
add!(sdh_porous_quad, :u, ipu_quad)
add!(sdh_porous_quad, :p, ipp_quad)
# Porous triangles
sdh_porous_tri = SubDofHandler(dh, getcellset(grid, "porous3"))
add!(sdh_porous_tri, :u, ipu_tri)
add!(sdh_porous_tri, :p, ipp_tri)

close!(dh)
```

DTU

Ferrite.jl + DifferentialEquations.jl



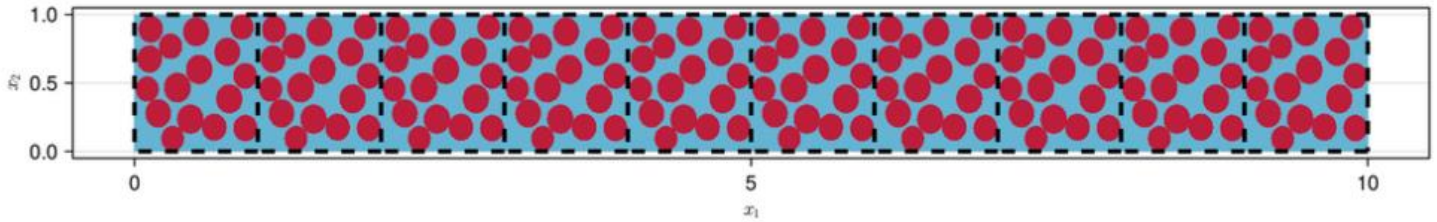
Some research examples

Homogenization of Structural Batteries

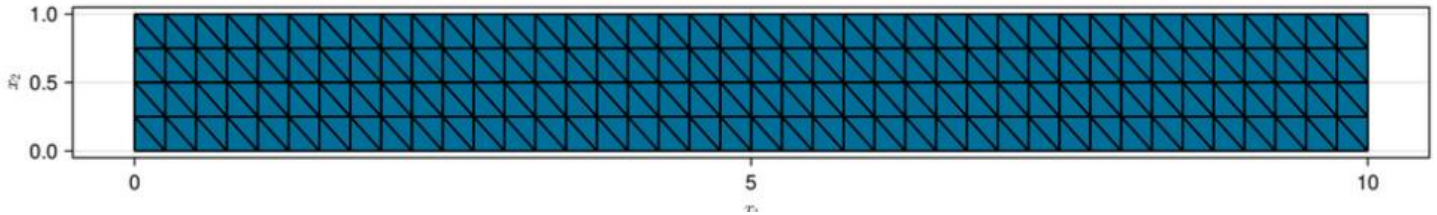
David Rollin: Institute of Applied Mechanics, TU Braunschweig

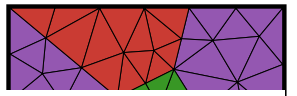
Modeling interfacial behavior in electroactive materials

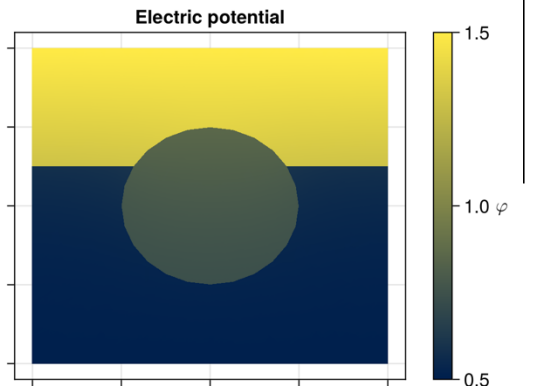
Direct Numerical Simulation (DNS) of an example problem

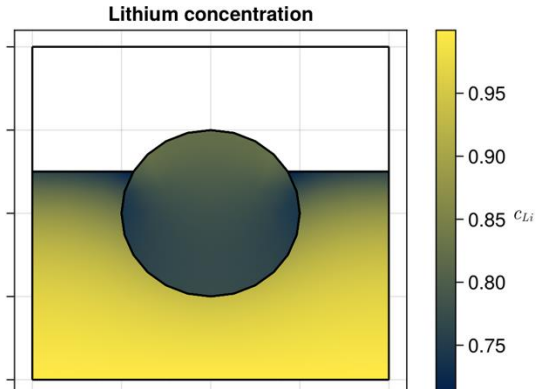


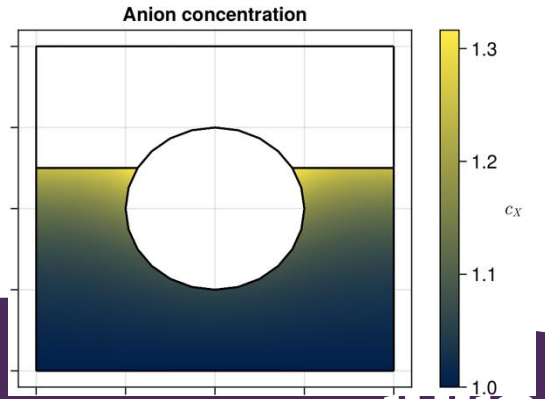
Corresponding macro-scale problem







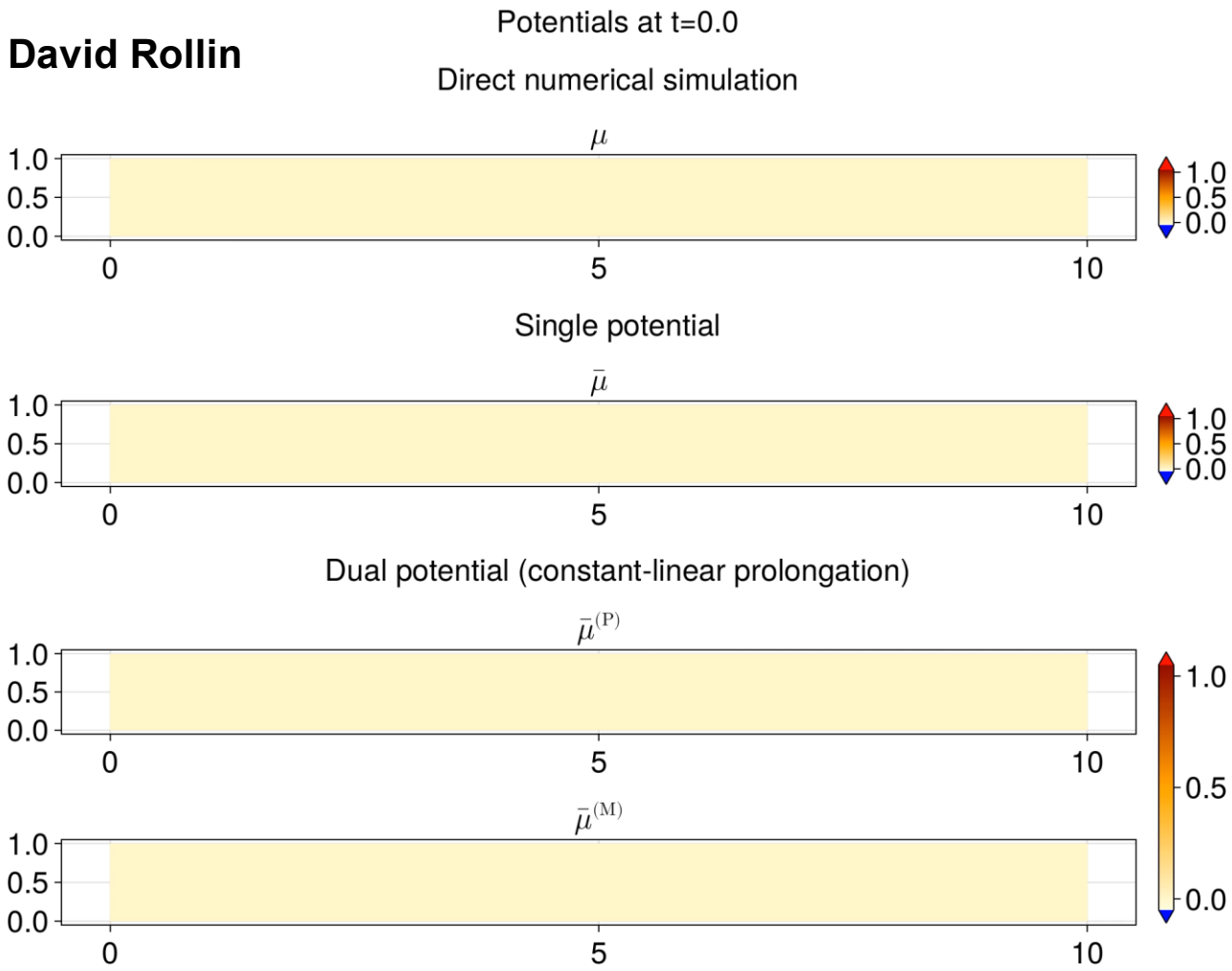


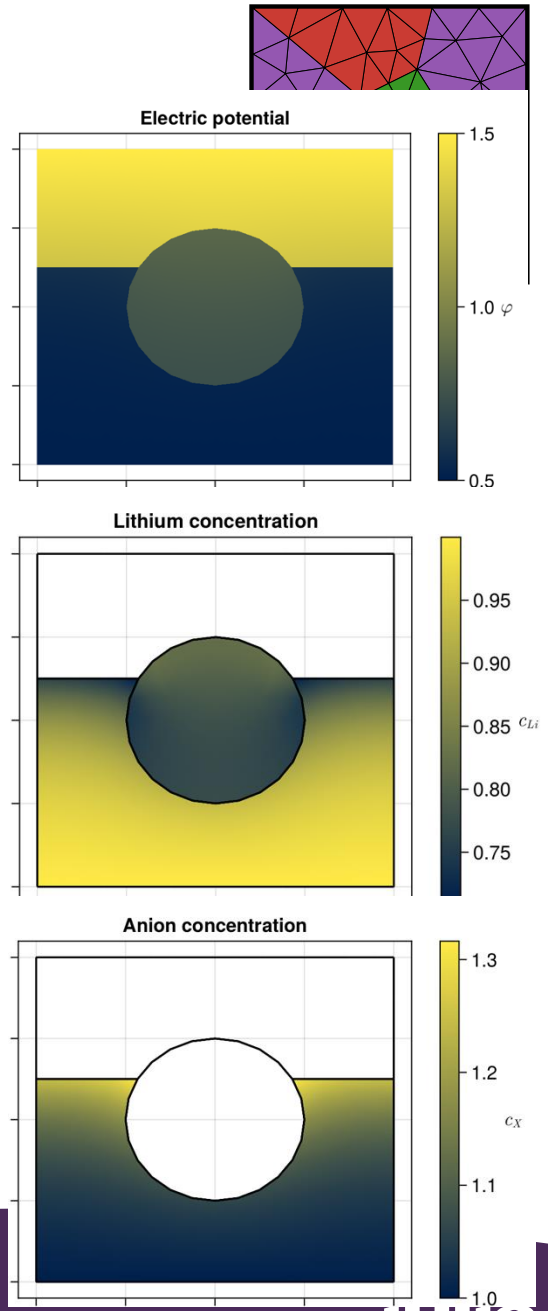


D. R. Rollin, F. Larsson, K. Runesson, and R. Jänicke, "Upscaling of chemo-mechanical properties of battery electrode material," *Int. J. Solids Struct.*, vol. 281, no. February, p. 112405, 2023,

Homogenization of Structural Batteries

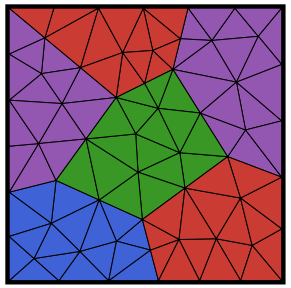
David Rollin





D. R. Rollin, F. Larsson, K. Runesson, and R. Jänicke, "Upscaling of chemo-mechanical properties of battery electrode material," *Int. J. Solids Struct.*, vol. 281, no. February, p. 112405, 2023,

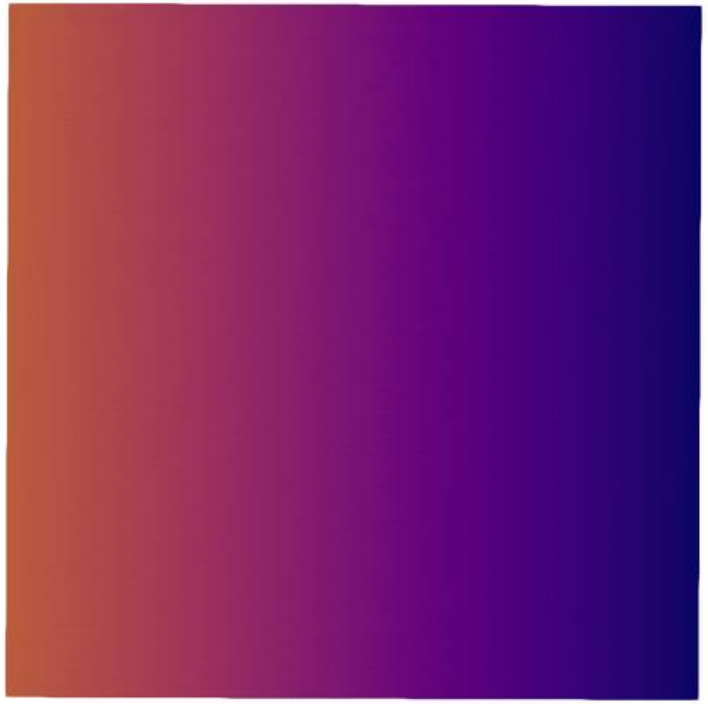
Cardiac Multiphysics

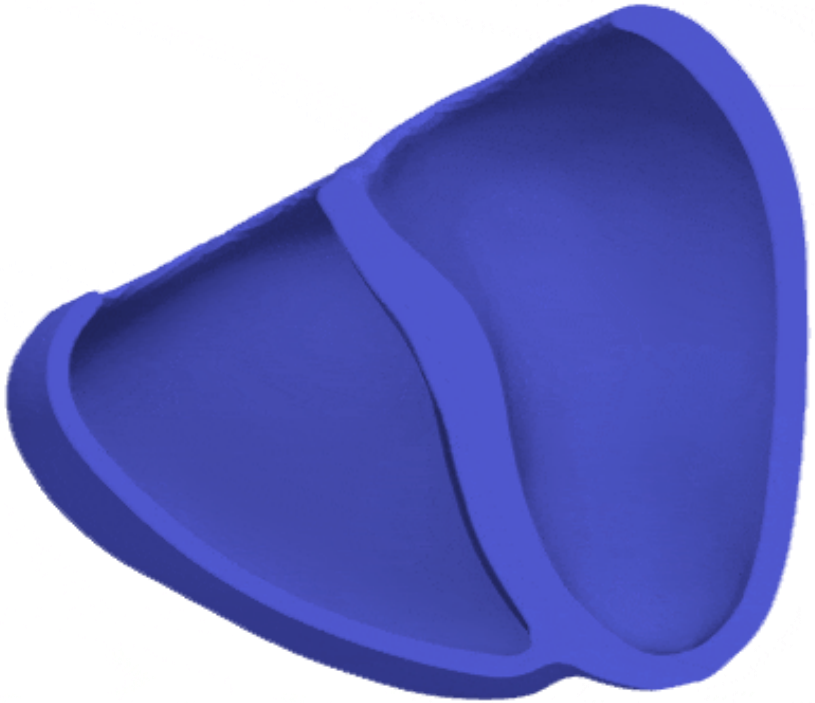


Dennis Ogiermann: Chair of Continuum Mechanics, Ruhr-Universität Bochum

Code:

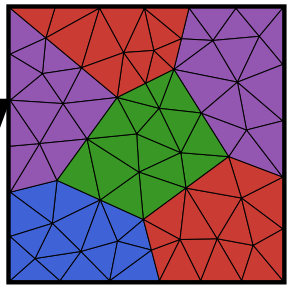
github.com/termi-official/Thunderbolt.jl



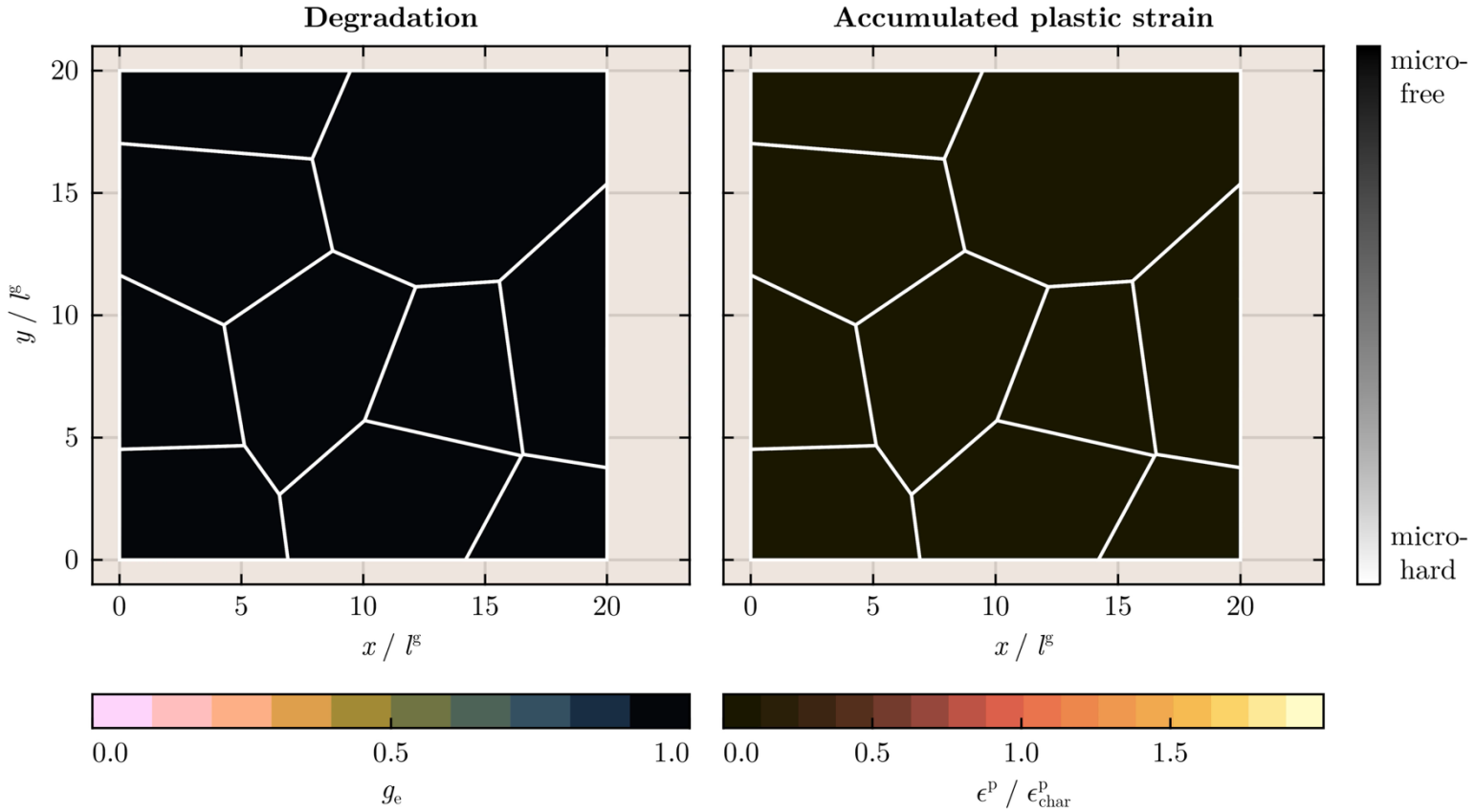


D. Ogiermann, D. Balzani, and L. E. Perotti, “An Extended Generalized Hill Model for Cardiac Tissue: Comparison with Different Approaches Based on Experimental Data,” in Functional Imaging and Modeling of the Heart, 2023, pp. 555–564.

Phase-field Fracture + Gradient Crystal Plasticity

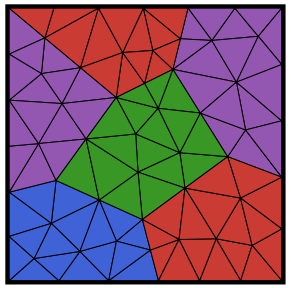


Kim Louisa Auth: Division of Material and Computational Mechanics, Chalmers, Sweden / Section of Solid Mechanics, DTU, Denmark

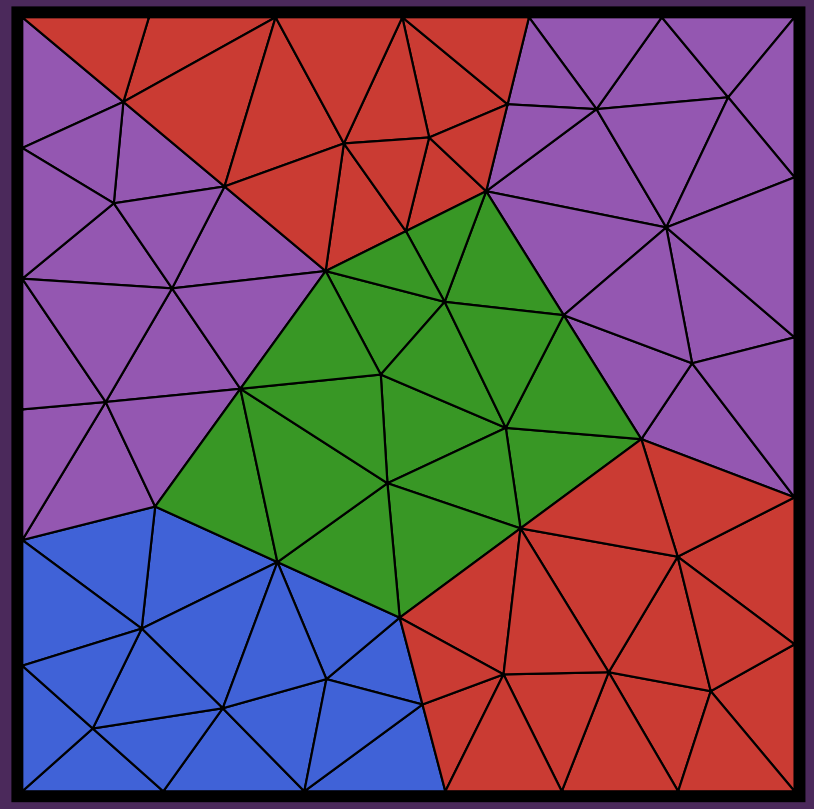


Auth, K. L., Brouzoulis, J., & Ekh, M. (2025). Phase-Field Modeling of Ductile Fracture Across Grain Boundaries in Polycrystals. *International Journal for Numerical Methods in Engineering*, *126*(12).

Community and documentation



- **Documentation** and examples: https://ferrite-fem.github.io/Ferrite.jl/
- **Slack**: https://julialang.org/slack/, and join #ferrite-fem
  - Getting help
  - Sharing code snippets
  - Discussion about solving problems, theory, etc.
- **Github**
  - Issues: Requesting features / reporting bugs
  - PRs: Making fixes / enhancements
  - Discussions: Asking questions



Ferrite.jl